# ARCHIVOS

# STREAMS

**Un stream es un buffer de almacenamiento de los datos que se envían o reciben en una operación de entrada o salida.**

**En formato binario no se produce ningún tipo de transformación de los datos**

**En el formato de texto se producen conversiones de los valores de los datos al formato ASCII.**

**Cuando sucede que el buffer se lleno se envía el contenido de este al dispositivo de salida.**

**En el caso de una operación de entrada se toman los datos hasta que se vacíe para realizar un nuevo ingreso de datos desde el dispositivo.**

**Este sistema de procesamiento de datos tiene la característica que cuando se produce un fin anormal del programa se pierden los datos almacenados en el flujo.**

# STREAMS STANDARD

**Existen 5 streams standars que se abren al iniciar un programa en C**

| | |
|---|---|
| **stdin** | **Teclado** |
| **stdout** | **Pantalla** |
| **stderr** | **Pantalla** |
| **stdprn** | **Impresora LPT1** |
| **stdaux** | **Puerto serie 1 COM1** |

# APERTURA DE ARCHIVOS

**FILE *fopen(char *nombre,char *modo)**

**nombre: nombre, unidad y ruta del archivo.**

**modo: forma de apertura del archivo.**

**Devuelve un puntero a FILE si es exitoso, si error devuelve NULL.**

| ATRIBUTO | SIGNIFICADO |
|---|---|
| r ó rt | Abre un archivo de texto para lectura. Si no existe producirá un error. |
| w ó wt | Crea un archivo de texto para escritura. En el caso de que exista destruye el contenido. |
| a ó at | Abre un archivo de texto para añadir. Si no existe crea uno, si existe agrega los datos al |

| ATRIBUTO | SIGNIFICADO |
|---|---|
| **rb** | **Abre un archivo binario para lectura. Si no existe producirá un error.** |
| **wb** | **Crea un archivo binario para escritura. En el caso de que exista destruye el contenido.** |
| **ab** | **Abre un archivo binario para añadir. Si no existe crea uno, si existe agrega los datos al final.** |
| **r+ ó r+t** | **Abre un archivo de texto para lectura/escritura. Si no existe no lo crea.** |
| **w+ ó w+t** | **Crea un archivo de texto para lectura/escritura. Si existe destruye los contenidos del archivo existente.** |
| **a+ ó a+t** | **Abre o crea un archivo de texto para lectura/escritura.** |
| **r+b** | **Abre un archivo binario para lectura/escritura. Si no existe no lo crea.** |
| **w+b** | **Crea un archivo binario para lectura/escritura. Si existe destruye los contenidos del archivo** |

# CIERRE DE ARCHIVO

**int fclose(FILE *fp)**

**fp : puntero asociado al archivo**

**Devuelve 0 si se ejecuta correctamente, si ocurre un error EOF.**

# ESCRITURA/LECTURA DE UN CARÁCTER

**int fgetc(FILE *fp)**

**fp: archivo asociado**

**fgetc devuelve el carácter si exitosa, EOF si error**

**int fputc(int ch,FILE *fp)**

**ch: carácter a escribir**

**fp: archivo asociado**

**fputc devuelve el carácter si exitosa, EOF si error**

# DETECCION DE EOF Y ERROR

**int feof(FILE *fp)**

**fp : archivo asociado**

**Devuelve distinto de 0 si EOF, en caso contrario 0**

**int ferror(FILE *fp)**

**fp: archivo asociado**

**Devuelve 0 si no hubo error, si hubo un error distinto de 0**

# LECTURA/ESCRITURA DE STRING

**int fputs(char *str,FILE * fp)**

**fp: archivo asociado**

**str: string en donde se ubica la cadena a escribir.**

**fputs devuelve EOF si error, en caso contrario distinto de 0**

**char *fgets(char *str,int num, FILE *fp)**

**fp: archivo asociado**

**num: cantidad de caracteres a leer, hasta ‘\n’ o EOF**

**str: string en donde se ubica la cadena a leer.**

**fgets devuelve str si es correcta, en caso contrario NULL**

# ENTRADA Y SALIDA CON FORMATO

**int fprintf(FILE *fp, char *control,**
**..................);**
**fp: archivo asociado**

**control: cadena de**
**control**
**.....................: lista de variables**

**Devuelve la cantidad de ítems escritos, si hubo error EOF**

**int fscanf(FILE *fp, char *control, ..................);**

**fp: archivo asociado**

**control: cadena de**
**control**
**.....................: lista de variables**

**Devuelve la cantidad de ítems leidos, si hubo error EOF**

# LECTURA/ESCRITURA DE BLOQUES

**size_t fwrite(void *buf, size_t size, size_t n, FILE *fp);**

**Escribe un bloque**

**size_t fread(void * buf, size_t size, size_t n, FILE * fp);**

**Lee un bloque**

**buf: puntero al bloque**

**size: tamaño del bloque**

**n: cantidad de bloques**

**fp: archivo asociado**

**size_t : unsigned int**

**Devuelve la cantidad de bloques, si hubo error un numero < n**

# DESPLAZAMIENTO

**int fseek(FILE *fp, long d, int origen);**

**fp: archivo asociado**

**d: desplazamiento**

**origen: posición de inicio del desplazamiento**

**Devuelve 0 si no hubo error, si hubo error distinto de cero**

| | |
|---|---|
| **SEEK_SET** | **comienzo del archivo** |
| **SEEK_CUR** | **posición actual** |
| **SEEK_END** | **fin del archivo** |

# DESPLAZAMIENTO

**long ftell(FILE *fp);**

**fp: archivo asociado**

**Devuelve la posición actual del archivo en bytes, si hubo error -1L**

# BORRAR UN ARCHIVO

**int remove(char *nombre);**

**nombre: unidad, ruta, nombre y extensión del archivo**

**Devuelve 0 si hubo éxito, si hubo error distinto de 0**

## RETORNA AL COMIENZO

**void rewind(FILE *fp);**

**fp: archivo asociado**

## LIBERAR EL BUFFER

**int fflush(FILE *fp);**

**fp: archivo asociado**

**Devuelve 0 si hubo exito, si hubo error EOF**

# ESTRUCTURA FILE

short level indica que tan vacío o lleno esta el buffer

signed flags banderas de estatus del archivo

har fd descriptor del archivo

unsigned char hold ultimo carácter leído en el buffer para utilizar ungetc

short bsize tamaño del buffer

signed char *buffer buffer de transferencia de los datos

signed char *curp posición actual del cursor del archivo

unsigned istemp indicador de archivo temporal

short token usado para chequeos de validación